



MASTERING THE

S⁺LC

A practical guide for creating success
with low-code for innovation


By Stephan Koning

Customer Success Lead at Betty Blocks

Table of contents

1. Speak up, spark change	6
Ensuring employee engagement and participation	7
Optimizing idea management processes	8
Providing supportive technology	9
2. Ideation	12
Initial screening	13
Scoring system	13
Tech stack compatibility assessment	15
Select the three best ideas	15
Provide feedback	18
3. Validation	20
User stories and acceptance criteria	20
Determine the appropriate delivery path	24
User story mapping	28
Visualize the user journey	30
Update the user story map	31
Finalize the business case, get the green light	31
4. Development	32
Controlling the project	32
Alignment between the business and it through fusion teams	35
Development best practices	37

5. Deployment	44
Deployment planning	44
Communication plan	47
User acceptance testing (UAT)	50
Celebrate!	55
6. Maintenance	56
Low code maintenance	56
Setting up maintenance processes	59
Allocating time and resources	61
Promote success	63
Conclusion	67



Introduction

The Software Development Life Cycle (SDLC) is the backbone of software development, encompassing ideation, validation, development, deployment, and maintenance. In the rapidly evolving digital landscape, each phase is critical in transforming an idea into a functional, scalable, maintainable, and widely adopted application. With the rise of low-code platforms like Betty Blocks, the traditional complexities of software development are significantly reduced as they empower organizations to streamline these phases, fostering innovation and reducing time-to-market. This book aims to be your roadmap to mastering the SDLC with low-code.

By democratizing application development, low-code platforms enable a broader range of contributors, from professional developers to business users. This inclusivity accelerates digital innovation, allowing organizations to rapidly respond to market demands and user needs. We refer to this as 'Citizen Development'. Citizen Development bridges the gap between technical and non-technical teams, enabling a more diverse group of innovators to participate in the development process. This not only accelerates the development cycle but also enhances the creativity and problem-solving capabilities within teams.

Innovation, low-code, and Citizen Development are deeply interconnected within the SDLC. Low-code platforms enhance the innovation process by reducing barriers to development, allowing more ideas to be rapidly prototyped and tested. Citizen Development empowers a wider range of stakeholders to contribute, ensuring that diverse perspectives are considered, which often leads to more innovative solutions.

However, flaws in a company's SDLC can lead to significant issues, including delays, budget overruns, security vulnerabilities, and subpar software quality. These problems can erode customer trust, reduce market competitiveness, and increase operational costs. Misalignment between business objectives and the final software product due to a flawed SDLC can cause inefficiencies and missed opportunities. This is why mastering the SDLC is so important: it ensures that development processes are efficient, predictable, and aligned with business goals.

In this book, we delve into each phase of the SDLC, providing practical insights and strategies tailored to low-code development. From initial ideation to continuous maintenance, by mastering the SDLC with a low-code approach, you can achieve greater efficiency, drive digital transformation, and ensure the long-term success of your applications.

1. Speak up, spark change

In the current business environment, size alone does not guarantee success. Companies that innovate and challenge the status quo tend to outpace even the largest competitors. Speed and agility, powered by modern technologies like low-code platforms, are key to staying ahead. Low-code enables rapid application development, embodying the essence of digital innovation by facilitating faster time-to-market.

The foundation of leveraging low-code for digital innovation success lies in engaging and motivating employees to contribute their innovative ideas. Creating an environment that encourages employees to speak up and challenge existing norms is crucial. This openness allows for the submission of ideas that can lead to exploring new horizons, enhancing productivity, boosting customer and employee satisfaction, and reducing costs. Too often, companies miss out on innovative ideas because they restrict input to specific teams, ignoring valuable insights from other departments. This can hinder the company's competitive edge, not due to a lack of technology but because of insufficient processes to highlight unique concepts.

Having cutting-edge technology is meaningless if there's nothing to build with it. There are three essential components for facilitating people and processes in gathering digital innovation initiatives: ensuring employee engagement and participation, optimizing idea management processes, and providing supporting technology.

Ensuring employee engagement and participation

Digital innovation thrives when employees are deeply engaged and actively participating. This engagement starts with transparent communication about the company's goals and ambitions. Employees need to understand the company's vision and their role in achieving it, creating a sense of urgency and intrinsic motivation. Clear, consistent communication fosters ownership and commitment. Additionally, employees must be aware of the expected innovative behaviors, which are typically embedded in the company's norms and values. Clear understanding and expectations are the pillars of establishing a culture of innovation.

Creating an environment where employees feel empowered to contribute is crucial. This involves not only encouraging them to share their ideas but also providing the necessary support and resources to bring those ideas to life. Establishing an innovation committee, representing various departments or expertise areas, to facilitate and oversee the entire process of submitting, evaluating, and implementing innovative ideas is an effective way to scale employee participation. Such a committee actively promotes innovation throughout the company and encourages cross-functional collaboration. For instance, creating cross-functional teams for brainstorming sessions allows for diverse perspectives, enriching the innovation process and boosting engagement. Additionally, implementing a formal rewards program to celebrate successful ideas and their contributors fosters a culture of innovation.

To ensure sustained engagement, it is vital to regularly update employees on the progress and impact of their ideas. Sharing success stories and publicly recognizing active contributors can motivate others to participate. Acknowledging efforts and demonstrating how employee input is making a tangible difference reinforces the importance of their contributions to the company's success.

Optimizing idea management processes

Engaging employees is just the first step; the next crucial component is optimizing the management of their ideas. An effective idea management process ensures that all innovative concepts are captured, evaluated, and acted upon systematically. This process starts with defining clear goals and objectives that align with the company's overall strategy. By providing a structured framework, employees can focus their creativity on areas that will deliver the most value.

A well-defined submission process and transparent evaluation criteria are essential for managing the flow of ideas. Clear and fair evaluation criteria maintain consistency and encourage participation. Offering a centralized place where employees can easily submit their ideas in a structured format ensures that no valuable input is overlooked and information is efficiently gathered for evaluation purposes.

Regularly reviewing and prioritizing submitted ideas ensures that the most promising concepts are developed further. It is important to provide feedback to employees on their submissions, whether or not their ideas

are selected for implementation. Constructive feedback helps refine future submissions and keeps employees motivated.

Organizing innovation events, such as hackathons or idea competitions, can also stimulate creativity and collaboration across the organization. These events provide a structured opportunity for employees to brainstorm and develop their ideas, fostering a sense of community and shared purpose.

Providing supportive technology

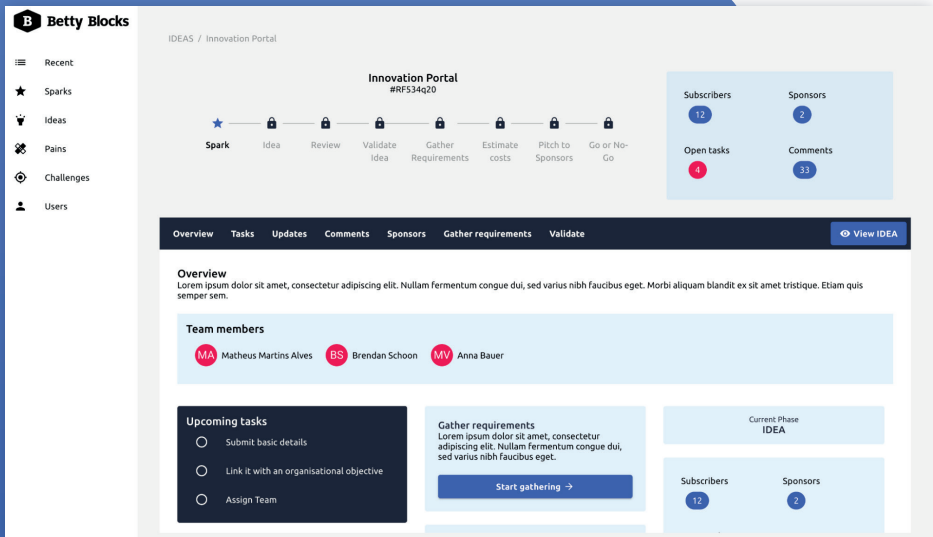
Sustaining digital innovation at scale requires robust technological support. Managing over 100 submitted ideas manually, such as in an Excel file, quickly becomes cumbersome and inefficient, increasing the risk of overlooking valuable concepts. This inefficiency also reduces the time available for communication, and the lack of feedback can demotivate employees from sharing their ideas.

Implementing a centralized portal for efficient idea management and collaboration is an excellent starting point. This portal should be accessible to all employees, offering a user-friendly interface for submitting, tracking, and managing ideas. It can also serve as a repository for success stories and best practices, providing inspiration and guidance for future projects. Incorporating basic data analysis capabilities within such a platform can help identify patterns and trends in the submitted ideas, enabling the organization to focus on the most promising innovations. Highlighting popular and impactful ideas increases visibility and encourages further participation, creating a virtuous cycle of continuous improvement and engagement.

If you're already using a low-code platform, why not build such a portal yourself? Figures 1 and 2 are examples of an Innovation Center of Excellence built with the Betty Blocks platform that guides users through the process, requiring them to submit a minimum set of information.

Bottom line: For digital innovation to thrive, people, processes, and technology must operate in harmony.

Figure 1: The Betty Blocks Innovation Center of Excellence workflow



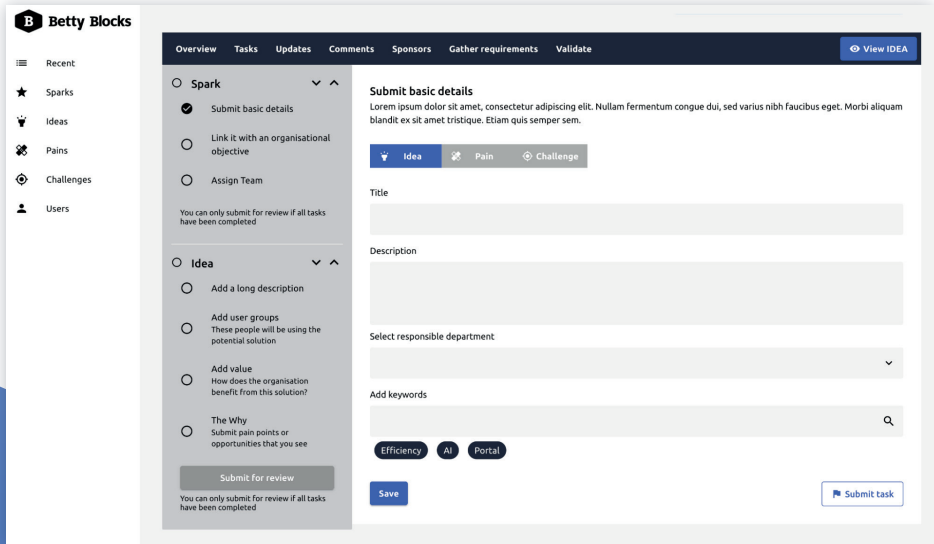


Figure 2: submitting basic details

2. Ideation

When applied successfully, the approach described in Chapter 1 generates a long list of innovative ideas. This is where the process of ideation begins: assessing the feasibility of the most promising ideas to deliver measurable value for the company. In other words, it involves refining a long list of ideas into a tangible shortlist of potential use cases suitable for application development.

Turning an idea into an innovative digital solution requires a sharp selection process, focusing on end user adoption and value creation. This step is crucial, especially since Citizen Development has democratized the skills required for delivering software applications. With low-code technology, it becomes increasingly easier for problem owners to become problem solvers. However, Citizen Developers, driven by enthusiasm, often start building immediately without properly validating the feasibility of their initiatives or identifying the requirements for the first application version (more on this in Chapter 3).

This is understandable, as developing software applications is not usually the full-time job of a Citizen Developer. They often seek fast solutions due to time constraints. Consequently, they may skip the crucial steps of assessing and validating ideas, relying on low-code platforms to build applications quickly. However, these shortcuts rarely outweigh the long-term problems they create, such as quality issues and poor end user adoption of the software application, the common reasons for failed Citizen Development projects.

An ideation strategy helps to mitigate these risks and is the starting point of turning ideas into impactful solutions.

Initial screening

Ideation begins with an initial screening of all submitted ideas to eliminate duplicates, impractical concepts, or those that clearly do not align with your organization's goals or capabilities. Make a first selection by grouping ideas that have a measurable impact on your organization's desired outcomes.

Scoring system

Secondly, implement a scoring system to evaluate each idea based on the established criteria. These criteria typically include:

- **Number of employee upvotes:** Incorporates the perspectives and insights of employees and determines which ideas receive the most support from them.
- **Potential impact on the organization's desired outcomes:** Ensures that ideas are assessed based on their alignment with the organization's strategic goals and objectives. This focus on impact helps prioritize ideas that contribute most significantly to organizational success.
- **Feasibility:** Assesses the practicality and achievability of each idea within the organization's resources and capabilities. Prioritizing feasible ideas with ideally the least amount of effort increases the likelihood of successful implementation and minimizes wasted effort on unrealistic proposals.

- **Resource requirements:** Helps the organization understand the resource implications of implementing each idea, enabling better resource allocation and planning. Clear insight into resource needs facilitates decision-making and budgeting processes.
- **Scalability:** Evaluates the potential for ideas to grow and adapt as the organization evolves, ensuring long-term viability and sustainability. Prioritizing scalable ideas allows the organization to pursue opportunities with lasting impact.
- **Complexity and time constraints:** Identifies the level of complexity associated with implementing each idea and assesses whether it can be accomplished within specified timeframes. This insight enables realistic project planning and management of time-sensitive initiatives.
- **Potential risks:** Highlights potential drawbacks and challenges associated with each idea, allowing organizations to proactively mitigate risks. Assessing risks helps prevent costly failures and ensures that decision-makers have a comprehensive understanding of the implications of pursuing each idea.

The evaluation can be conducted using various scoring methods:

- **Numerical scores:** Assign a numerical score to each idea for each criterion based on a predetermined scale (e.g., 1 to 10). The scores can be aggregated to provide an overall assessment of each idea.
- **Rankings:** Rank each idea relative to others based on how well they meet each criterion. For example, rank ideas from most to least feasible, impactful, etc.
- **Yes/no assessment:** Determine whether each idea meets each criterion with a simple yes or no. This approach provides a binary evaluation of each criterion.

Tech stack compatibility assessment

At this stage, it's crucial to determine whether the highest-scoring ideas can be developed within the company's existing technology systems or if a solution already exists. For example, at Betty Blocks, a platform fit calculator facilitates this process by answering the question: buy or build? This type of questionnaire provides advice based on the answers given, indicating whether the idea aligns with the platform's capabilities and, if so, which building and non-building skills are required to successfully deliver the software application.

It's important to investigate whether a solution already exists and can be reused, if a standard off-the-shelf solution is available for purchase, or if the solution requires customization that would be better built in-house using low-code technology.

Select the three best ideas

Based on the scoring system and technology compatibility, you now have a shortlist of feasible ideas that are suitable for application development. It is now time to get the right people in the room: the idea initiators and your stakeholders. Work with them on capturing the business value the shortlisted ideas intend to produce. This results in a tangible deliverable: a compact business case per idea that allows you to select the three best ideas, present those to management, and ultimately select the winning idea for application development. As part of your innovation strategy, consider all of the above steps as an iterative process in order to build up a roadmap of valuable use cases.

If idea initiators and stakeholders are conducting ideation sessions with the goal of building a solution themselves, provide them with the necessary instructions and tools to achieve the desired outcome.

Capturing the intended business value requires effective ideation techniques. At the beginning of your innovation journey, start with the two most common techniques:

Brainstorming workshop

- Brainstorming workshops are structured sessions where a group of individuals comes together to generate creative ideas and solutions for a specific problem or challenge.
- Participants typically follow a set of guidelines to encourage free thinking and idea generation. This often involves suspending judgment, encouraging quantity over quality, building on others' ideas, and aiming for a wide variety of suggestions.
- A facilitator usually leads the session, guiding participants through the brainstorming process and ensuring everyone has an opportunity to contribute. To stimulate creativity, the facilitator may use techniques such as mind mapping, word association, or visual aids.

Hackathon

- Hackathons are intensive events where individuals or teams collaborate intensively on software or hardware projects, often within a limited timeframe, to create functioning prototypes for solutions.
- Participants typically gather for a day or two days to work on projects. They may start with brainstorming sessions to generate project ideas and then form teams based on shared interests or skills. Throughout

the event, participants work on designing and building their prototypes.

- Hackathons often provide a collaborative and creative environment, with access to mentors, resources, and sometimes prizes for winning projects.

Whether you go for a brainstorming workshop or a hackathon, the following outputs of such sessions are required at minimum:

Target group

- Application users and roles
- Pain points and opportunities per user group

Application components

- Business requirements
- Integrations
- (happy) Workflow (figure 3)

Alignment with the organization's desired outcomes/strategic initiatives

- Measurable value
- Success factors

This information is best captured in a Solution Value Board (figure 4), which provides a high-level overview of the intended solution. For those new to ideation, a lightweight approach can be taken initially by simply listing requirements in a Business Requirements Document (figure 5). Both tools can easily be built with low-code, enabling companies to facilitate and manage ideation processes within their existing technology stack.

With this information, a ballpark cost estimation of the project, including future maintenance costs, can be provided in a compact business case.

Provide feedback

Document the rationale behind evaluating each idea and communicate the decisions made, along with the reasons behind them, to keep idea submitters (your potential Citizen Developers) engaged. Ideally, this should be done through the use of an Innovation Center of Excellence, as mentioned in Chapter 1.

ID#	REQUIREMENT NAME	DESCRIPTION	CATEGORY	REQUESTER	PRIORITY	OUT OF SCOPE	NOTES
1	User Authentication	Users should be able to register an account and log in securely using email and password.	Security	IT	High	<input type="checkbox"/>	Ensure compliance with industry security standards
2	Dashboard Analytics	Provide users with interactive dashboards to visualize key metrics and performance indicators.	Analytics	Marketing	Medium	<input type="checkbox"/>	Prioritize customizable dashboard layouts and support for exporting data
3	Mobile Compatibility	Ensure that the application is accessible on mobile devices, including smartphones and tablets.	User Experience	Business	Low	<input checked="" type="checkbox"/>	Ensure testing to be done cross devices

Figure 5: Business Requirements Document



End goal / Value driver

What drives the organisation to build this solution

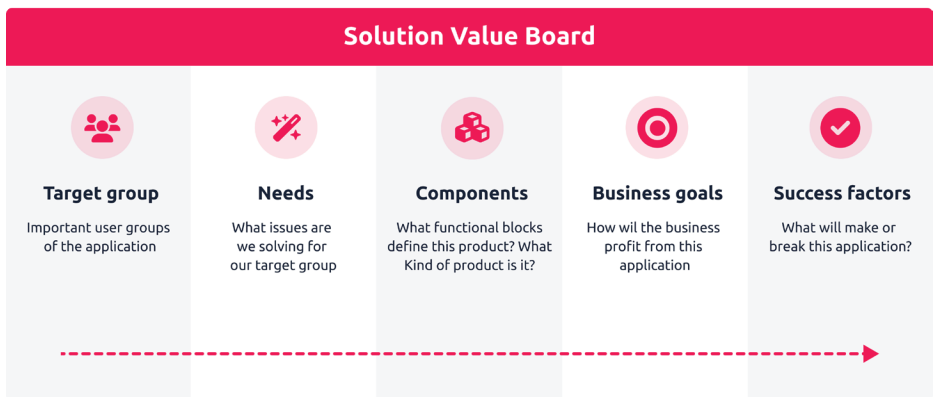
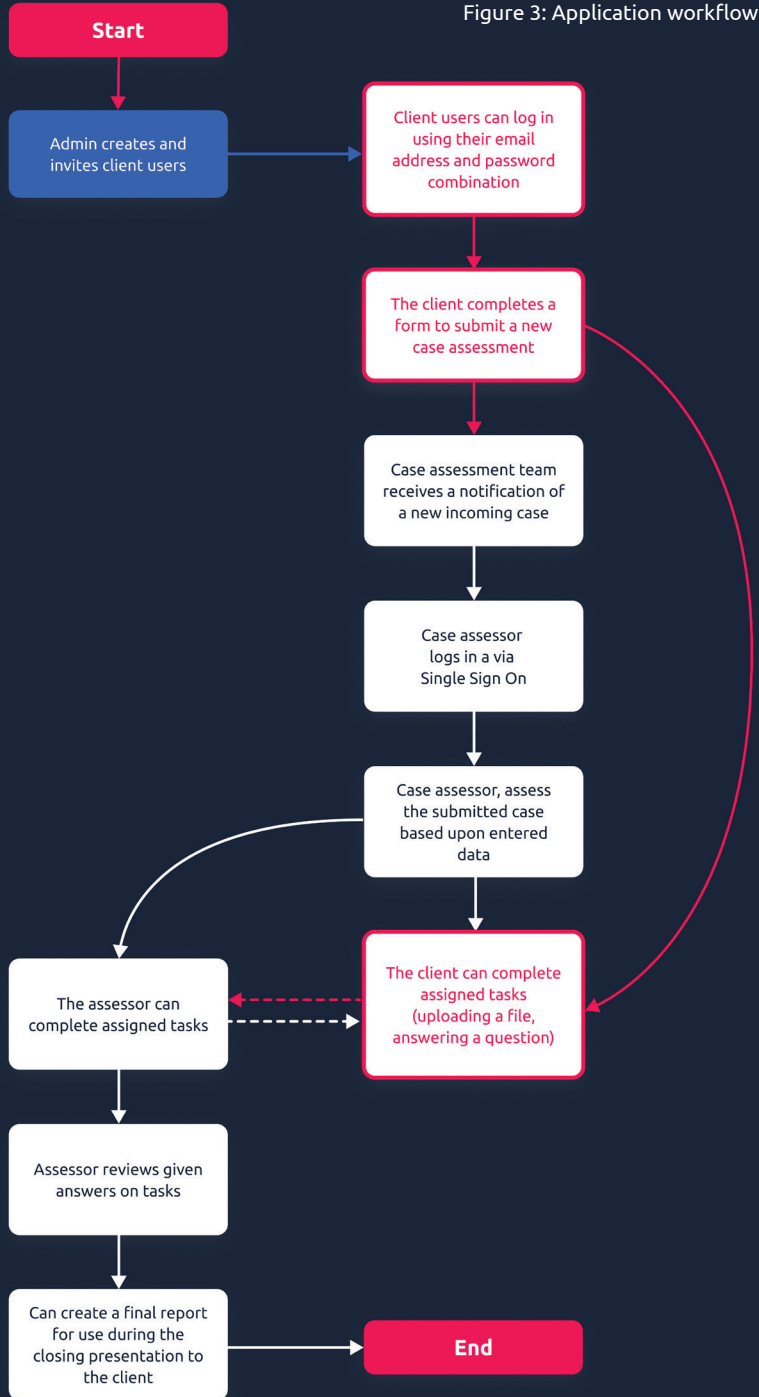


Figure 4: Solution Value Board

Figure 3: Application workflow



3. Validation

During the ideation Phase, the focus was on collecting business requirements, whereas the Validation Phase entails gathering application requirements, commonly referred to as ‘scoping’ activities. It’s the stage where the proposed use case undergoes close inspection to ensure it aligns with the needs and objectives of stakeholders before moving forward into application development. The extent of scoping and validation depends on the complexity and risk of the project versus the skills needed to execute it. A thorough understanding of key concepts is crucial before developing a validation strategy.

User stories and acceptance criteria

A user story is a concise, informal description of a feature or requirement from the perspective of an end user. It’s a way for development teams to capture and communicate what needs to be built into a software project in a simple and understandable format.

Typically, a user story follows a specific template: “As a [type of user], I want [some goal] so that [some reason].”

For example: “As a website visitor, I want to be able to create an account so that I can access exclusive content.”

User stories are intentionally brief and high-level, serving as placeholders for conversations between the development team and stakeholders.

They provide context and direction for the development process without prescribing specific technical solutions. User stories are given more detail by acceptance criteria, which are specific conditions or requirements that must be met for a user story to be considered complete and satisfactory. They outline the functional and non-functional expectations of the feature or functionality described in the user story. Clear acceptance criteria help minimize misunderstandings and ensure that the delivered product meets the desired quality standards.

An example of acceptance criteria for a user story typically looks like this:

User Registration Form (Functional Requirement):

- The website should have a registration form with fields for the user's name, email address, and password.
- The registration form should include the following validation for the email address format: minimum two characters@minimum two characters.domain extension.

Performance (Non-Functional Requirement):

- The registration and account activation processes should be completed within 30 seconds.
- The login process should have a response time of less than two seconds.

When brainstorming innovative ideas during the ideation phase, you've already identified a potential workflow. There are various ways to turn that workflow into a visual user journey:

UI Mock-up

Static screen designs that give an impression of the look and feel of the application.

Wireframe

Visualizes the user journey of the application by focusing on the functional components and where to place them on the pages. There are two different types of wireframes:

- Low fidelity (figure 6): Simple, basic sketches that outline the structure and functionality of a design without detailed content or style.
- High fidelity (figure 7): High-fidelity designs are detailed, polished representations that closely mimic the final product, including visuals, typography, and interactions.

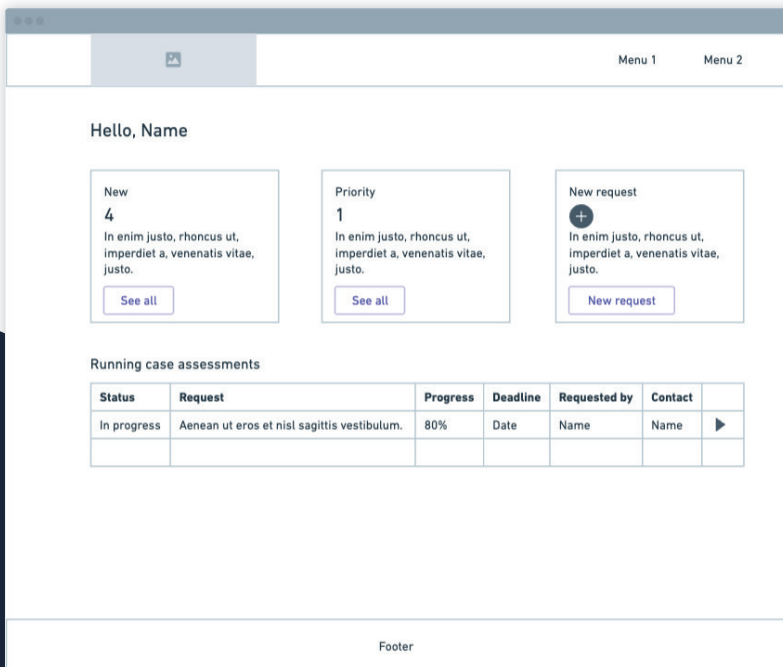


Figure 6: Low fidelity

Hello, Brendan Schoon

New

4

Pellentesque id eros vulputate, ornare orci eget.

→ View New

Priority

1

Etiam ut velit augue. Vestibulum ornare consequat odio nec scelerisque

→ View Priority

New request

+

Nam hendrerit enim ut nulla ornare, a venenatis mauris placerat.

→ New request

Running case assessments

Request	Progress	Deadline	Requested by	Contact
<p>In progress</p> <p>Aenean ut eros et nisi sagittis vestibulum. Vivamus consectetur hendrerit lacus.</p>	<p>80%</p>	<p>7 days left</p> <p>in 30-07-2024</p>	R. Baker	H. Hagen
<p>In progress</p> <p>Aenean ut eros et nisi sagittis vestibulum. Vivamus consectetur hendrerit lacus.</p>	<p>80%</p>	<p>7 days left</p> <p>in 30-07-2024</p>	R. Baker	H. Hagen
<p>In progress</p> <p>Aenean ut eros et nisi sagittis vestibulum. Vivamus consectetur hendrerit lacus.</p>	<p>80%</p>	<p>7 days left</p> <p>in 30-07-2024</p>	R. Baker	H. Hagen
<p>In progress</p> <p>Aenean ut eros et nisi sagittis vestibulum. Vivamus consectetur hendrerit lacus.</p>	<p>80%</p>	<p>7 days left</p> <p>in 30-07-2024</p>	R. Baker	H. Hagen
<p>In progress</p> <p>Aenean ut eros et nisi sagittis vestibulum. Vivamus consectetur hendrerit lacus.</p>	<p>80%</p>	<p>7 days left</p> <p>in 30-07-2024</p>	R. Baker	H. Hagen

Figure 7: High fidelity

- **Prototype:** An early version of the application that contains working functions and is used to test the UI in action, as well as to analyze the effect on the user.

Determine the appropriate delivery path

Validation begins with assessing the complexity and risks involved in the use case to determine the appropriate delivery path. This path outlines the necessary scoping activities and required skills to deliver the application. Naturally, a low-complexity, low-risk use case requires fewer scoping efforts than a high-complexity, high-risk use case.

Factors influencing complexity and risks:

Complexity

- **User Interaction:** The complexity of user interactions and interfaces required by the use case.
- **Functionality:** The number and complexity of functions or custom-coded features the use case demands.
- **Data:** The complexity and size of data structures, sources, and flows involved in the use case.
- **Integration(s):** The level of integration required with other systems, APIs, or services.
- **Business Logic:** The sophistication of the business rules and logic embedded within the use case.
- **Non-Functional Requirements:** Performance, scalability, security, and usability, which can add layers of complexity.

Risks

- Security: Potential breaches of confidentiality, data loss, and uncontrolled user access.
- Business Criticality: The ability of the business to continue operations if the application fails.
- IT Architecture: Rules for building, modifying, and interfacing IT resources.
- Regulatory: External laws, industry standards, or compliance requirements.
- Reputational: Potential loss of brand perception, social capital, or market share.
- Financial: Potential impact on revenue.

As part of Betty Blocks' Citizen Development vision, three different delivery paths are applicable:

1. The Citizen Developer path
2. The Business Technologist path
3. The Low-Coder path

The Citizen Developer path

A Citizen Developer is a business user with expertise in his/her business domain. They're natural problem solvers looking for ways to improve their way of working with digital solutions with as little effort as possible. A Citizen Developer mostly works from pre-defined no-code templates built on a low-code platform and configures them to his/her needs.

Complexity of the use case

Low, pre-defined templates have restricted (governed) configuration options.

Required skills

Basic platform introduction on how to configure templates and business logic knowledge.

Scoping activities

As the template usually covers 90% of the requirements in terms of design and functionality, listing down the business requirements for the remaining 10% is sufficient. It is far more efficient for a Citizen Developer to simply start configuring rather than spending too much time gathering requirements of functionalities that are easily built.

The Business Technologist path

A Business Technologist is a tech-savvy business user dedicated to streamlining and automating business processes, improving efficiency, and reducing manual workload. The Business Technologist closes the gap between the business and IT by creating no-code solutions that address the unique needs and challenges by translating creative business concepts into practical applications.

Complexity of the use case: Low - Medium

Low: The application is not available in a template and must be built from scratch. However, it has multiple reusable components, allowing a Business Technologist to deliver the application rapidly.

Medium: The application must be built from scratch and requires advanced business logic workflows and/or integrations with other systems.

Required skills

Advanced low-code skills and business logic knowledge.

Scoping activities

Several deep dive sessions, also referred to as 'Refinements' are conducted to capture the user stories, acceptance criteria, and user journeys, which are translated into wireframes.

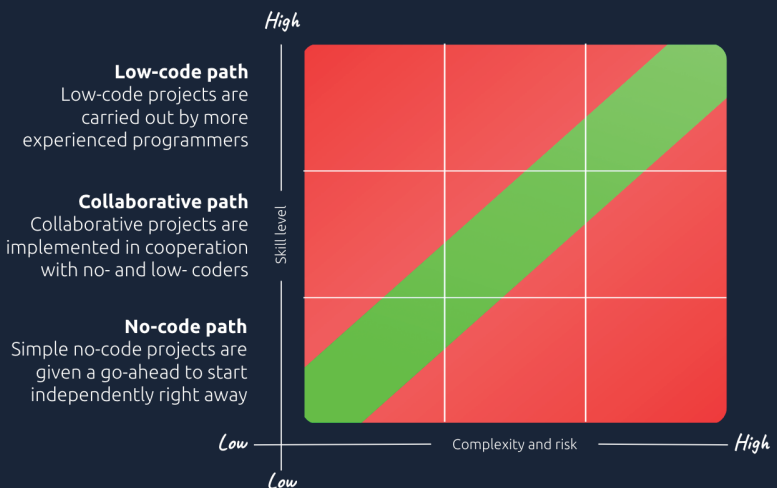


Figure 8: The sweet spot of use cases fit for Citizen Development

The Low-Coder path

A Low-Coder is a professional software developer skilled in coding. They use their skills to extend low-code platforms by customizing components, interfaces, data sources, and actions, which can then be reused by the entire company and/or built into highly complex applications.

Complexity of the use case

High: The application has to be built from scratch and requires highly advanced business logic workflows, integrations with other systems, and customizations in code.

Required skills

Mastery of different coding languages as well as advanced data modeling.

Scoping activities

Several refinements to capture the user stories, acceptance criteria, and user journeys, which are translated into a prototype. Technical research in order to determine the viability of the use case is conducted through a Proof of Concept.

User story mapping

Once the appropriate delivery path is determined, it's time to gather the user stories. User story mapping is a scoping technique used in software development to describe and prioritize the features or functionalities of a product from the perspective of its users. It helps teams understand the user's journey through the product and ensures that the development process aligns with the user's needs and goals.

Identify user activities

The team starts by refining the business requirements captured in the Solution Value Board. They then list all the activities that each user might perform within the product.

Create user stories

For each activity identified, the team creates user stories. It is highly recommended that these be added and managed in software tools such as Jira. Use the INVEST method to map them:

- **Independent:** User stories should be self-contained and independent of each other, meaning they can be developed, tested, and delivered without relying on other stories.
- **Negotiable:** User stories should be open to discussion and negotiation between the development team and the stakeholders.
- **Valuable:** Each user story should deliver value to the end user or customer. It should address a specific need or provide a meaningful benefit.
- **Estimable:** User stories should be clear and concise enough to allow the development team to estimate the effort required to implement them.
- **Small:** User stories should be small enough to be completed within a single iteration or sprint. They should focus on delivering a small, manageable piece of functionality that can be implemented and tested within a short timeframe.
- **Testable:** User stories should be written in a way that allows their acceptance criteria to be easily defined and tested.

For medium to complex use cases, aim for quick and to-the-point refinements. It's better to do multiple rounds of iterations than to spend too long on creating user stories that might need changing after feedback.

Visualize the user journey

Determine the appropriate design technique based on the chosen delivery path and begin by mapping the user journey, arranging the user stories horizontally in the order of the workflow they typically follow. This creates a timeline or sequence of user interactions with the product and serves as the input for the design.

With low-code technology, Citizen Developers can participate in the design process using user-friendly drag-and-drop capabilities. This allows them to rapidly visualize their solutions and obtain feedback from end users and stakeholders. Additionally, designs are created directly within the development environment, significantly speeding up the delivery process compared to building designs in a separate system.

The crucial part of this process is the actual validation of the use case design with end users and stakeholders to gather early feedback. This not only gains support but also helps identify any overlooked factors, preventing rework costs and quality issues. Design validation ensures that everyone has a shared and accurate understanding of the intended use case outcome and that it accurately represents their requirements. This phase is essential for laying the groundwork for future adoption.

Update the user story map

Use the feedback collected from end users and stakeholders to revise the user story map. New user stories may be added and outdated ones removed. After validation, incorporate acceptance criteria into the user stories. Continuously translate feedback into new or updated designs, iterating until you've defined the scope of the Minimum Viable Product (MVP).

Finalize the business case, get the green light

Finally, complete your business case by incorporating a project quotation that considers estimated costs of user stories, necessary skills and resource availability for application delivery, and projected maintenance costs and efforts. Calculate Return on Investment (ROI) and identify relevant metrics for monitoring adoption and value. Following approval of your business case, secure management commitment to assemble your team and start the project!

4. Development

By now, the ideal use case has been selected, validated with stakeholders and end users, and the necessary skills and resources have been identified for project delivery. It's time to start the building process. To ensure the application meets the desired quality standards and is scalable for future enhancements, it's essential to establish project control mechanisms, assemble a fusion team, and incorporate design and development best practices into the building process.

Controlling the project

While low-code technology has made software development more enjoyable and accessible to a wider audience, it remains inherently complex, especially when creating something entirely new. In such cases, assumptions are often made due to the lack of existing models for comparison.

The well-known Agile approach offers a solution. By working in short, focused iterations known as 'Sprints' and quickly delivering each iteration to stakeholders and end users, Agile enables informed decisions based on real-world feedback. Adjusting your plan based on early and valuable feedback is incredibly beneficial, as it reveals exactly what is needed to gain user support, while minimizing both time and cost implications.

However, what if the iterative Agile process isn't feasible due to budget constraints or limited time and resources? The following project control mechanisms help address such constraints while still maintaining Agile principles:

Conduct a Proof of Concept first

Estimates in software development are hard to rely on. They generally serve as indications meant for project budget requests and are based on what you 'know' up front. But what do you really know? When you are building something that doesn't exist, you can't know what you don't know, so everything is assumption-based. Due to the complex nature of software development, unexpected complexities will arise. To mitigate these risks, consider conducting a Proof of Concept during the Validation phase. This involves delving into the most complex and challenging aspects of your project upfront to gain insights and anticipate potential roadblocks. By addressing these complexities early on, you'll be better equipped to make accurate estimates and navigate the development process more effectively.

Scope statement

While feedback is valuable, not all suggestions are relevant to the Minimum Viable Product (MVP). It's essential to carefully assess which proposed changes align with the MVP scope and which don't. Incorporating changes beyond the MVP scope can lead to scope creep and budget overruns, which can be harmful to the project's success. One effective way to manage this is by utilizing a scope statement, typically included in a Statement of Work. This document outlines all the user stories, acceptance criteria, and wireframes identified for the MVP. The budget is based on estimates derived from this scope statement. Formal agreement on the scope and budget with stakeholders can provide clarity and alignment. When there's

budget flexibility, implementing feedback-driven changes becomes feasible. However, if budget constraints arise, the scope statement serves as a tool for collaborative decision-making on functionality trade-offs. Stakeholders could also decide to allocate additional budget for specific functionalities if necessary. Regardless of the outcome, it's crucial to document all changes to track how the extra budget is utilized. This ensures transparency and accountability in budget management and allows for reporting on budget allocation.

Prioritize features

Prioritize features based on complexity and their importance and value to the end user. Focus on delivering the most complex and critical functionalities first. This ensures that even if the budget becomes tight later on, you've already delivered the most essential aspects of the application.

Transparent communication

Maintain open and transparent communication with stakeholders by providing regular progress updates and demonstrations. At the end of each Sprint, concisely outline what has been delivered compared to the budget expended. Transparency is key, especially when aspects of the project take longer than anticipated. This openness fosters trust and enables stakeholders to grasp the project's current status and any budgetary limitations. Well-informed and involved stakeholders are more inclined to allocate additional budget when necessary.

Continuous improvement

Embrace a culture of continuous improvement within the team. Encourage team members to identify inefficiencies and suggest ways to optimize processes to deliver more value. It goes without saying that a motivated

team, focussed on continuous improvement is a high-performing, cost-effective team.

Alignment between the business and IT through fusion teams

When the Citizen Developer path (Chapter 3) is the best delivery path for the use case, Citizen Developers can handle it solo by tweaking a pre-made template. It's akin to using a tried-and-true recipe that already meets the organization's Non-Functional Requirements.

However, the Business Technologist or Low-Coder paths involve greater complexity. These paths require starting from scratch, making it impractical for one person to handle everything from gathering requirements to testing and managing stakeholders.

This is where fusion teams come in. These teams are composed of individuals from diverse backgrounds and skill levels who collaborate to achieve specific business goals. Figure 9 illustrates how different developer personas collaborate, each bringing their expertise, experience with the low-code platform, and teamwork skills to the table. Fusion teams prioritize toolkits and rely on pre-built components and templates to accelerate development while ensuring consistency, standardization, and scalability.

With business-savvy Citizen Developers and Business Technologists teaming up with IT's Low-Code Developers and Platform Admins, alignment with IT is crucial for fusion teams to succeed. The old-school approach of assigning all tech tasks solely to IT is no longer effective.

Modern organizations integrate business and tech goals, fostering creativity and innovation. To ensure fusion teams operate smoothly while adhering to security, compliance, and best practices, establishing a Center of Excellence is essential. This serves as a central hub for expertise, guidance, rules, and a space to share knowledge and ideas, making low-code development safe to scale across the organization.

Figures 10, 11, and 12 illustrate how the Betty Blocks Innovation Center of Excellence functions not just as a portal to submit and manage innovation ideas, but as a centralized place to service a low-code community of practice.



Figure 9: Collaboration in a fusion team example of fusion team

Figure 10: A landing page with the latest new innovation center of excellence

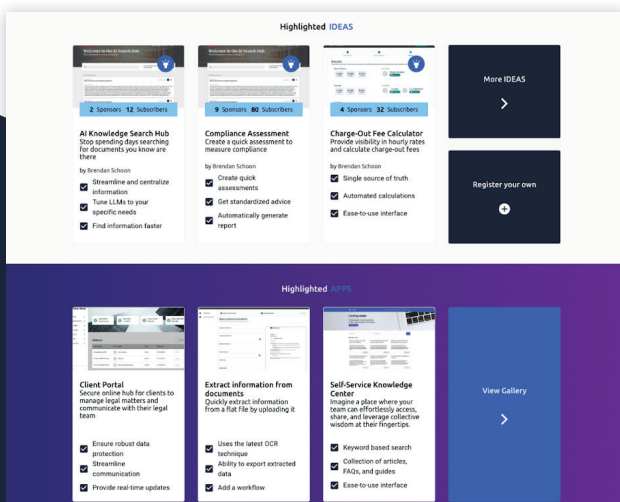
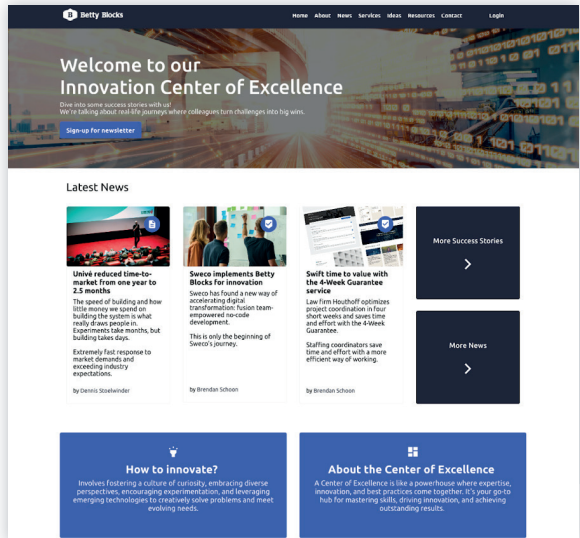


Figure 11: Highlighted ideas and apps ideas and apps

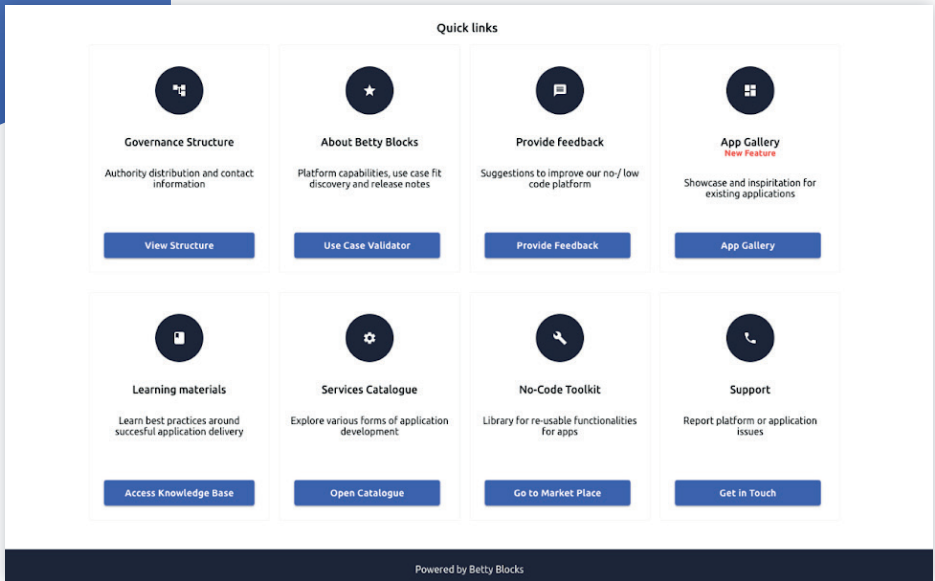


Figure 12: Quick links to relevant information extra information

Getting a fusion team up and running smoothly, with everyone collaborating effectively, is key to delivering a top-notch product. But it's just as important to stick to best practices when building a software application. Spending that extra time upfront on quality design always pays off in the end, whether it's in boosting user adoption or saving on costs by avoiding rework due to bugs.

Here's a breakdown of the most common low-code development best practices:

Modularize your application

- Break down your application into smaller, reusable components or modules. Modular design not only improves maintainability and

scalability but also allows for easier reusability across different parts of your application or even in future projects.

Follow design standards

Responsive design

Ensures that a website or application adapts seamlessly to different screen sizes and devices, providing an optimal viewing and interaction experience for users across desktops, tablets, and smartphones.

Consistent navigation

Ensures that users can easily navigate through the application and find what they need without confusion. This includes maintaining a consistent placement of navigation elements such as menus, buttons, and links across different pages or sections of the application.

Consistent branding and visual style

Consistent branding and visual style across the application help establish a strong brand identity and foster user recognition and trust. This includes using consistent colors, typography, imagery, and design elements throughout the application to reinforce brand personality and values.

Error prevention and recovery

Designing with error prevention and recovery in mind helps minimize user frustration and errors. This involves implementing clear and descriptive error messages that explain the issue and provide guidance on how to resolve it. Additionally, offering undo options, confirmation dialogs for critical actions, and checkpoints in forms or workflows can help users recover from mistakes easily and continue their tasks without interruption.

Feedback and confirmation

Providing feedback and confirmation to users after their actions helps reduce uncertainty and enhances the user experience. This includes displaying visual cues such as animations, tooltips, progress indicators, or success/error messages to indicate that an action has been completed successfully or to alert users of any issues or errors.

Optimize performance

- Pay attention to performance considerations throughout the development process. Optimize your low-code applications for speed, responsiveness, and efficiency. This includes minimizing unnecessary data fetching, optimizing database queries, and implementing caching mechanisms. Performance tuning should be an ongoing effort, with regular monitoring and optimization to ensure optimal application performance.

Test early and often

- Ensure the quality of your software by thoroughly testing for functionality, usability, security, and performance, meeting the expectations of your users. Daily deployments from development to specific testing environments are essential to prevent environmental inconsistencies.
- Integrating test cases into user stories eliminates ambiguity about when a user story is considered 'Done.' Adhering to this quality-by-design principle ensures that team members follow structured

processes and verify new functionalities before they reach end users. Typically, test cases are crafted by team members with specialized testing expertise and usually look like this:

“As a customer, I want to purchase a product online so that I can conveniently buy items from the comfort of my home.”

Scenario 1 - Happy flow (successful purchase)

1. Open the shopping website.
2. Search for the desired product.
3. Add the product to the shopping cart.
4. Proceed to checkout.
5. Enter valid shipping and billing information.
6. Complete the purchase.
7. Verify the order confirmation.

Scenario 2 - Error flow (product out of stock)

1. Open the shopping website.
2. Search for the desired product.
3. Verify the product is out of stock.
4. Attempt to add the out-of-stock product to the shopping cart.
5. Verify the error message about the product being out of stock.

Facilitate flow-based testing for end users. By providing them with testing scripts (figure 13) that navigate through various application flows instead of solely assessing individual user stories, you enrich their overall navigation experience. Flow-based testing offers insights into how functionalities synergize within real-world usage scenarios. Ensure these tests are

conducted in every Sprint, with a focus on regression testing (continuously test what has already been delivered) to prevent any reemergence of previously resolved issues.

Document your work:

- Maintain documentation that is easily accessible and understandable to team members, stakeholders, and future developers. Clear and comprehensive documentation not only facilitates knowledge sharing and collaboration but also aids in troubleshooting, maintenance, and future enhancements.
- Ensure documentation is updated regularly, ideally during each Sprint, to minimize the required effort.

Flow	Instructions
Front End url	
Back Office url	
Test Date	
Registration and login	In the backoffice, go to settings > webusers
Registration in backoffice	Press 'New'
Login via Front-end	Fill in all information and in roles choose 'webuser' and press save
	As webuser press the link in the received email
	Fill in a password and press 'Change'
	Press on 'Back to login'
	Log in with email address and password

Figure 13: Testing script format requirements check test script

Expected result	OK/NOK	Feedback
The overview with webusers is visible	OK ▾	
A form is shown that enables you to create a new webuser	NOK ▾	No form visible
A new webuser is created. The webuser receives an email to set up their password	▾	
A frontend page with the url app-identifier/reset-password/:unieke_code is shown	▾	
You see the notification 'Password Changed'	▾	
You see the page app-identifier/login	▾	
You see the page app-identifier/home	▾	

5. Deployment

While deploying an application is a moment of celebration, it is also a delicate process, as you only get one chance to make a first impression. Successful deployment requires concrete planning, a solid communication strategy, and a comprehensive training and support plan to ensure that end users have everything they need to use the application effectively.

Deployment planning

As deployment is a delicate process, it requires planning for ensuring a smooth and successful transition from development to production. Deployment planning provides a structured approach that helps identify and mitigate potential risks. A well-crafted deployment plan outlines clear steps, timelines, and roles, ensuring that all team members understand their responsibilities, and tasks are executed efficiently. By anticipating challenges and preparing contingency plans, deployment planning enhances the overall reliability and performance of the application, ultimately leading to higher user satisfaction and business continuity.

The deployment process begins with defining the scope of the deployment. This step involves outlining the components to be deployed, the systems that will be affected, and the anticipated impact on the organization. Next, the environment is assessed and prepared. Evaluating the current infrastructure is crucial to ensure it can support the new application. Additionally, it's important to verify that backup systems are in place and functioning correctly to prevent data loss.

With the environment prepared, a detailed deployment plan is developed. The pre-deployment phase includes several key steps. First, conduct a configuration review to ensure all configurations are correct and consistent across environments. Following this, perform pre-deployment testing, and conduct User Acceptance Testing in a staging environment that mirrors the production environment. It's also essential to communicate with users, informing stakeholders about the deployment schedule, expected impacts, instructions, and support availability.

The deployment phase follows, where the deployment, according to the predefined steps is executed, which includes software installation, configuration, and data migration. Initial checks, such as sanity checks and smoke testing, are performed to validate that the deployment is successful.

After deployment, post-deployment steps are critical. Thorough testing is conducted to ensure the application functions correctly in the production environment. Monitoring tools are set up to track application performance and detect any issues. Gathering initial user feedback helps identify any immediate problems.

Creating a timeline overview is a vital part of the deployment plan. This timeline outlines key milestones and deadlines for each phase of the deployment. During the pre-deployment phase, tasks and deadlines for the preparation stage, such as configuration reviews and pre-deployment testing, are defined. The deployment phase specifies the exact timing for deployment activities, ensuring minimal disruption to business operations, preferably outside business hours. The post-deployment phase sets timelines for post-deployment testing, monitoring, and user feedback collection.

Assigning roles and responsibilities ensures a smooth deployment process:

- The project manager oversees the entire deployment process, ensuring tasks are completed on time, objectives are met, and managing communication with stakeholders.
- The product owner facilitates User Acceptance Tests and post-deployment testing, acting as a liaison between the development team and the business.
- The deployment team lead manages the technical aspects of the deployment, coordinating with developers, testers, and system administrators.
- Developers are responsible for preparing the application for deployment, fixing any last-minute issues, and supporting the deployment process.
- Testers conduct pre-deployment and post-deployment testing to ensure the application functions as expected.
- The support team provides assistance to users during and after the deployment, handling queries and troubleshooting issues.

Risk management and contingency planning are essential to address potential issues that could impact the deployment. Identifying risks, such as technical issues, resource limitations, or unforeseen business disruptions, helps in developing mitigation strategies. Contingency plans, including rollback procedures, are prepared for critical scenarios in case the deployment fails.

The actual deployment execution and monitoring involve using a deployment checklist to ensure all steps are completed correctly and in the right order. Real-time monitoring tracks system performance, user activity,

and any emerging issues. An immediate response plan is in place to respond quickly to any issues that arise during the deployment, minimizing downtime and disruption.

Finally, a post-deployment review and continuous improvement process is conducted. A review meeting with key stakeholders is held to discuss what went well and what could be improved. Feedback is collected from users and team members to identify areas for improvement. Documentation is updated to reflect lessons learned and any changes made to the deployment process. The insights gained from the review and feedback are used to improve future deployments, ensuring a more efficient and effective process.

Communication plan

A communication plan is vital during the deployment phase to ensure that all stakeholders are informed, prepared, and aligned. It helps manage expectations by clearly outlining the deployment schedule, potential impacts, and any changes or new features users should anticipate. Effective communication minimizes confusion and anxiety, provides users with the necessary support and resources, and facilitates quick resolution of any issues that arise. By keeping everyone in the loop, a communication plan ensures a smoother transition, reduces the risk of disruptions, and enhances overall user satisfaction and adoption of the new application.

Creating a communication plan begins with identifying stakeholders. Internal stakeholders include project team members, IT staff, management, and

other departments that may be affected. External stakeholders include end users, customers, clients, and partners who will feel the impact of the deployment.

Next, it is essential to define objectives. One primary goal is to inform stakeholders by clearly communicating the deployment schedule, potential impacts, and new features or changes. Managing expectations is equally crucial, as well as setting realistic anticipations about the deployment process, including possible downtime and performance improvements. Providing support ensures stakeholders know how to access help and resources during the transition.

Developing key messages is the subsequent step. The deployment schedule must specify the date and time of the deployment, along with any expected downtime or maintenance windows. It is important to explain the purpose and benefits of the deployment, highlighting the reasons behind it and the anticipated advantages, such as new features, bug fixes, or performance enhancements. The impact on users should be described, detailing how the deployment will affect them, including any changes to workflows, potential disruptions, and preparation steps. Information on support and resources should be provided, indicating how users can get help if they encounter issues during or after the deployment.

Choosing appropriate communication channels is vital for the effective distribution of information. Email can be used to make detailed announcements and updates to all stakeholders. Information should also be posted on the company's intranet or internal portal for easy access. Meetings and webinars can be hosted to explain the deployment process and address any questions or concerns. Pop-up notifications within

applications can inform users about the deployment and any immediate actions they need to take. Detailed documentation, including FAQs, user guides, and troubleshooting tips, should be available on internal portals or help desks.

Creating a communication timeline ensures stakeholders are kept informed throughout the deployment process. In the pre-deployment phase, an initial announcement should notify stakeholders about the upcoming deployment a few weeks in advance, providing an overview of what to expect. Reminder notices should be sent one week and one day before the deployment to reinforce key messages and ensure everyone is prepared. During deployment, real-time updates on the progress should be provided, especially if there are any unexpected issues or delays. Post-deployment communication involves sending a deployment completion notice to inform stakeholders once the deployment is complete and the system is live, highlighting any immediate actions they need to take. Follow-up communication a few days after the deployment gathers feedback, addresses any ongoing issues, and provides additional support.

Developing a feedback loop is essential for continuous improvement. Surveys and feedback forms can be created to gather input from users about their experience with the deployment. Clear channels for user support, such as a dedicated helpdesk, email support, or chat service, should be established for reporting issues and asking questions. A post-deployment review meeting with key stakeholders should be conducted to discuss the deployment's success and areas for improvement.

Finally, roles and responsibilities must be assigned. A communication lead should be appointed to oversee the communication plan and ensure all

messages are clear and consistent. Subject matter experts (SMEs) should be identified to provide detailed information and support for users during the transition. The support team must be prepared to handle an increase in inquiries and issues during and after the deployment.

User acceptance testing (UAT)

UAT is the final phase of the software testing process, where the application is tested by its intended audience in a real-world environment. Even though an application undergoes rigorous testing throughout the development process, this broader round of testing involves a larger and more diverse audience, helping to identify any issues that might not have been apparent in earlier stages. It ensures that the application meets the needs and expectations of all end users, verifying that it performs well under real-world conditions. This comprehensive testing helps catch last-minute bugs, validates the user experience, and builds confidence in the application's readiness for full-scale deployment.

The process begins with thorough planning and preparation. The first step is to define the objectives clearly. This involves outlining the goals of User Acceptance Testing (UAT), such as validating the functionality, usability, and performance of the application. Once the objectives are set, selecting participants becomes essential. A representative group of end users from different departments or user segments is chosen to participate in the testing. To ensure comprehensive testing, creating test cases based on real-world scenarios and business processes that the application is expected to handle is crucial.

With the planning complete, the next phase is setting up the environment. It is important to establish a production-like environment that closely mirrors the actual production environment to ensure accurate results. Data preparation follows, where real or realistic test data is used to simulate actual usage conditions, providing a realistic testing scenario.

The execution of tests is where the participants come into play. They execute the test cases, interacting with the application as they would in their daily tasks. During this phase, documentation is key. Any issues, bugs, or unexpected behaviors encountered during testing are carefully recorded.

Feedback collection is the subsequent step. Detailed feedback from participants on their experience is gathered, including any difficulties or suggestions for improvement. Surveys, questionnaires, or interviews are employed to collect qualitative feedback on user satisfaction and usability, providing valuable insights.

Issue resolution follows the feedback collection. The identified issues are prioritized based on their severity and impact on business processes. Critical bugs and issues uncovered during UAT are addressed by working closely with the development team.

Verification and validation are crucial to ensure the effectiveness of the fixes. Retesting the affected areas after issues are resolved is necessary to confirm that the fixes are effective and have not introduced new problems. Obtaining formal sign-off from key stakeholders and participants is the final step in this phase, confirming that the application meets the acceptance criteria and is ready for deployment.

The plan concludes with documentation and reporting. A test summary report is prepared, summarizing the UAT process, findings, issues resolved, and the overall readiness of the application. User documentation is also updated based on feedback received during UAT, ensuring that user manuals and support materials are current and helpful.

Training and support plan

A training and support plan is crucial during the deployment phase to ensure users can effectively transition to and utilize the new application. Proper training equips users with the knowledge and skills needed to navigate the new system, reducing the learning curve and minimizing operational disruptions. Comprehensive support ensures that any issues or questions are promptly addressed, enhancing user confidence and satisfaction. By providing user manuals, tutorials, and accessible helpdesk services, a training and support plan fosters smooth adoption, maximizes productivity, and ensures the deployment's overall success.

The first step in the training process involves identifying the training needs of various user segments. Different user segments are determined based on their roles and responsibilities, and training resources are tailored to meet the specific needs of each group, whether they are end users, administrators, or technical staff. Assessing current skill levels helps identify gaps that need to be addressed for effective use of the new application.

With the training needs identified, the next phase is developing training materials. Comprehensive user manuals are created, covering all aspects of the application, including installation, configuration, and daily usage. These manuals feature step-by-step instructions with screenshots or illustrations

to ensure they are easy to follow. A FAQ section is incorporated to address common questions and issues users might encounter. Additionally, video tutorials are produced to demonstrate key features and common tasks within the application, using clear and concise language along with visual aids. Interactive tutorials are also developed, allowing users to practice using the application in a simulated environment. Live webinars are hosted to provide an overview of the application, demonstrate its functionality, and answer user questions in real-time. Quick reference guides, such as cheat sheets and infographics, are created to summarize essential functions and visually represent workflows and key processes within the application.

Providing access to these training resources is crucial for user engagement. A centralized portal or learning management system (LMS) is set up, where users can access all training materials. A searchable knowledge base is developed, including user manuals, tutorials, and FAQs for easy retrieval of information. Ensuring that all training resources are accessible to users with disabilities is a priority, achieved by providing alternative formats such as text descriptions for images and closed captions for videos.

Scheduling training sessions is another important aspect of the process. Pre-deployment training sessions are conducted to familiarize users with the new application, offering multiple sessions to accommodate different time zones and schedules. Post-deployment training sessions are also provided to reinforce learning and address any issues that arise after the deployment.

Offering ongoing support is essential to ensure users can fully utilize the application. A dedicated helpdesk support team is established to assist users with any issues or questions during and after deployment. Multiple channels for support are provided, including email, phone, live chat, and an online

ticketing system. On-demand support is facilitated through an up-to-date knowledge base that is accessible at any time and chatbots that provide instant answers to common questions and guide users to relevant resources. Peer support is encouraged through user forums or discussion boards where users can share experiences, ask questions, and help each other. Mentorship programs pair less experienced users with power users or mentors who can provide guidance and support.

Collecting feedback and measuring the effectiveness of the training is vital for continuous improvement. Surveys are distributed after training sessions to gather feedback on the quality and effectiveness of the training materials and sessions. Usage analytics are monitored to identify areas where users may be struggling and need additional support or training. Establishing a feedback loop helps continuously improve training materials and support services based on user input.

Finally, updating training materials regularly is necessary to ensure they remain current and effective. Version control is maintained to ensure that users always have access to the most up-to-date information. Training materials are regularly updated based on feedback, new features, and changes in the application, ensuring continuous improvement and relevance.

Celebrate! 🎉

The innovative idea selected during the ideation phase has now been transformed into a deployed and validated solution. This milestone is worth celebrating. Take time to acknowledge and appreciate the contributions of everyone involved, and gather lessons learned to apply to future projects. Although the steps for successful application adoption have been thoroughly followed, ongoing monitoring is crucial to address any issues that arise and to identify opportunities for further improvement.

6. Maintenance

Think of your application like a car. If something breaks, it needs immediate fixing. But you don't want to wait until you're stranded on the roadside. Instead, you schedule regular maintenance to catch issues before they become problems. However, maintaining an application is even more complex. It's like not only keeping your car running smoothly, but also upgrading it constantly with new features, adapting it to evolving terrains, and ensuring it meets all the latest safety standards. It's a dynamic ride that requires constant attention and innovation.

Enter application maintenance, the most underestimated part of the Software Development Life Cycle. Maintenance policies and processes are required in order to make sure your application continues to deliver the intended business value.

Low code maintenance

There are four types of maintenance in the context of low-code development:

Corrective - Fixing errors and bugs.

Adaptive - Adjusting the software to changes in the environment.

Perfective - Improving the performance or maintainability of the software without changing its functionality.

Preventive - Looking ahead and preventing potential future issues.

Corrective maintenance

Maintaining applications built on a low-code platform involves a comprehensive approach to ensure their functionality, performance, and adaptability in a dynamic business environment. One crucial aspect is corrective maintenance, which focuses on fixing defects and bugs that are discovered after the application is deployed. Despite the robust nature of low-code platforms, issues can still arise due to unforeseen user interactions or environmental changes. Corrective maintenance is essential to address these problems promptly, ensuring the application remains reliable and efficient for end users.

Adaptive maintenance

Equally important is adaptive maintenance, which involves making modifications to the application in response to changes in the external environment. This could include updates to comply with new regulations, compatibility with new hardware or software, or changes in the business process(es). The flexibility of low-code platforms allows for rapid adaptation, enabling businesses to stay agile and responsive to market demands. Adaptive maintenance ensures that the application continues to meet the evolving needs of the organization without extensive redevelopment.

Perfective maintenance

In addition to adapting to external changes, perfective maintenance is vital for enhancing the application's functionality and performance. This type of maintenance involves refining the application to improve user experience, optimize performance, and add new features based on user feedback. With the iterative nature of low-code development, perfective maintenance can be seamlessly integrated into the development lifecycle, allowing for continuous improvement and innovation.

Preventive maintenance

Lastly, preventive maintenance plays a proactive role in the long-term health of the application. This involves regular updates and improvements to prevent potential issues before they occur. Preventive maintenance includes activities such as code refactoring, updating dependencies, and performance tuning. By anticipating and mitigating potential problems, preventive maintenance helps to maintain the application's stability and performance, reducing the likelihood of future disruptions.

Examining these four types of maintenance, it becomes clear that maintaining applications built on a low-code platform offers significant advantages over traditional coded application maintenance, especially in terms of speed, ease of use, adaptability, and scalability:

Speed

Low-code platforms streamline and accelerate the maintenance process. The visual development interface and pre-built components allow for rapid updates and bug fixes, significantly reducing the time required for corrective and adaptive maintenance compared to traditional coding environments.

Ease of use

Low-code platforms are user-friendly and accessible to individuals without extensive programming knowledge. This democratizes maintenance, enabling a broader range of team members to participate in the process, reducing reliance on specialized developers, and allows for direct business participation and collaboration.

Adaptability

Low-code platforms are designed for flexibility, allowing applications to

be easily modified to meet changing business requirements. This simplifies adaptive maintenance, enabling quick updates and integration with other systems, unlike traditional coded applications, which often require more extensive changes.

Scalability

Low-code platforms offer scalable solutions that can grow with the business. New features and functionalities can be added without extensive redevelopment, making it easier to scale applications compared to traditional coded applications, which might require significant modifications to achieve the same scalability.

Setting up maintenance processes

Fortunately, there's already an excellent maintenance framework out there for effectively managing, supporting, and continuously improving applications: The Information Technology Infrastructure Library (ITIL). These are the key ITIL best practices to consider in the context of low-code development when setting up maintenance processes:

Incident management (corrective maintenance): Restore normal service operation as quickly as possible to minimize business impact.

- Establish a process for logging, categorizing, prioritizing, and resolving incidents.
- Use a centralized (digital) service desk for reporting and managing incidents.
- Implement escalation procedures for complex or high-priority incidents
- Monitor and review incident trends to identify recurring issues.

Problem management (preventive maintenance): Identify and eliminate the root cause of incidents to prevent recurrence.

- Conduct root cause analysis for significant or recurring incidents.
- Maintain a known error database to document identified problems and workarounds.
- Implement proactive problem management to identify potential issues before they occur.
- Collaborate with other teams to resolve underlying problems.

Change management (adaptive maintenance): Control the lifecycle of all changes to minimize disruption to IT services.

- Implement a change management process and applicable tooling for requesting, assessing, approving, and reviewing changes.
- Maintain a change schedule to coordinate the timing of changes.
- Ensure thorough testing and documentation of changes before implementation.

Release and deployment management: Ensure that new or changed services are released and deployed successfully.

- Develop a release management process for planning, building, testing, and deploying releases.
- Use automation tools for deployment to reduce errors and increase efficiency.
- Conduct post-release reviews to identify areas for improvement.

Service level management: Ensure that IT services meet agreed-upon service levels and business expectations.

- Define and document service level agreements (SLAs) with business

stakeholders.

- Monitor and report on service performance against SLAs.
- Regularly review and update SLAs to reflect changing business needs.
- Implement corrective actions to address any service-level breaches.

Continuous improvement (perfective maintenance): Continuously improve IT services and processes to add value to the business.

- Implement a continuous service improvement (CSI) process to identify and prioritize improvement opportunities.
- Use key performance indicators (KPIs) and metrics to measure service performance.
- Conduct regular reviews and assessments to identify areas for improvement.
- Foster a culture of continuous improvement within the IT organization.

Allocating time and resources

Given that resources are often scarce and capacity is under pressure, balancing issue resolution, problem prevention, and enhancement implementation in an application requires a well-structured approach. When starting a low-code project, the development team is usually also responsible for maintaining the application. Imagine the application becomes a big success, generating excitement and necessitating the launch of a new project. Suddenly, there's an "Error 404: Team Not Found".

Balancing productivity within a 40-hour work week requires acknowledging that only 20-25 hours are typically spent on delivering intended work. The remaining time often goes to unplanned tasks, setbacks, social interactions, and other distractions, which is completely normal. The key is to plan for this when scheduling work. The most efficient approach is to allocate time and work effectively during a timeboxed period, such as Sprints (Chapter 4):

- **50%** » Reserved for planned work items such as new functionalities for existing or new applications, bugs, or removing technical debt.
- **30%** » Reserved for unplanned tasks, such as unforeseen events and (high/critical priority) issues.
- **20%** » Reserved for non-productive time or “waste,” acknowledging inevitable distractions.

This approach allows for realistic planning of work while leaving room for unforeseen events. If these events don't occur or there are fewer bugs, the reserved time can be reallocated to planned work.

At a strategic level, it is beneficial to divide work topics into distinct buckets. This allows for better allocation of time and resources for planned work. Since end users are the primary users of the application, their feedback is crucial for delivering value. Keeping them engaged and satisfied is paramount. However, it is impossible to meet all their needs while ensuring the application remains future-proof and maintains high-quality standards. To address this, a similar approach can be applied that helps to balance immediate user needs with the long-term sustainability and quality of the application:

- **60%** » Reserved for implementing enhancements to existing applications or developing new applications requested by end users or stakeholders (perfective maintenance).
- **30%** » Reserved for ensuring the long-term health of applications (preventive and adaptive maintenance).
- **10%** » Reserved for various initiatives for the low-code team.

Promote success

Finally, ensure the success of the application is properly promoted. This is not just about sharing learnings and inspiring others but also about keeping your end users and stakeholders engaged. While it's easy to get caught up in new, exciting projects and overlook this step during the maintenance phase, it is crucial for several reasons:

1. **Sustained support and funding:** Stakeholders who see the tangible benefits and successes of the application are more likely to continue supporting it financially and administratively. Their backing is essential for securing ongoing resources, funding for new features, and overall project sustainability.
2. **Increased user adoption and engagement:** Engaged stakeholders can act as champions for the application within their departments, encouraging more users to adopt and utilize it effectively. Their positive outlook and endorsement can drive higher usage rates.
3. **Feedback and continuous improvement:** Engaged stakeholders are more likely to provide constructive feedback and innovative ideas for improvement. They can highlight areas where the application excels

and where it needs enhancements, ensuring it remains relevant and effective.

4. **Alignment with business goals:** Regular communication and showcasing of the application's successes help ensure that its development aligns with broader business objectives and strategic goals. Engaged stakeholders can guide and refine the application's direction to better serve the organization's needs.
5. **Mitigating resistance to change:** Engaging stakeholders and keeping them informed helps mitigate resistance to change. When stakeholders understand the benefits and see positive outcomes, they are more likely to support new initiatives and changes brought by the application.
6. **Enhancing organizational agility:** An engaged stakeholder group that regularly interacts with the application can quickly identify and respond to new business opportunities and challenges. This proactive stance allows the organization to remain agile and competitive.

Engagement ensures the application remains a valuable and dynamic asset to the organization. Keeping everyone engaged can be achieved in various ways:

Communicate success stories

- **User testimonials:** Collect and share positive feedback from end users who have benefited from the application. Highlight specific stories that demonstrate how the application has improved their workflow or productivity (Chapter 4).
- **Case studies:** Develop detailed case studies showcasing how different departments or teams are using the application to achieve their

goals. Include metrics and before-and-after scenarios to illustrate the impact.

Showcase metrics and analytics

- Usage statistics: Share usage statistics regularly, such as the number of active users, frequency of use, and key features being utilized. Use visual aids like graphs and charts for clarity.
- Impact metrics: Highlight key performance indicators (KPIs) that show the application's impact on business processes. This could include time saved, cost reductions, error rate reductions, or any other relevant metrics.

Organize demonstrations and workshops

- Live demonstrations: Conduct live demonstrations of the application for stakeholders, showcasing new features, improvements, and success stories. Interactive sessions where stakeholders can ask questions and provide feedback can be very engaging.
- Workshops and training: Offer workshops and training sessions to help stakeholders understand the application's capabilities and how it can further benefit their teams. This also helps in gathering more constructive feedback.

Use visual and interactive content

- Infographics and videos: Create engaging infographics and short videos that highlight the application's successes and key metrics. Visual content can be more impactful and easier to digest than long reports.
- Interactive dashboards: Develop interactive dashboards that

stakeholders can use to see real-time data and analytics about the application's usage and impact.

Celebrate milestones and achievements

- **Recognize key achievements:** Publicly recognize and celebrate key milestones and achievements related to the application's adoption and success. This can be done through internal announcements or even small events.
- **Rewards and recognition:** Implement a rewards and recognition program for teams or individuals who have significantly contributed to the application's success. This not only motivates but also acknowledges their efforts.

Foster a feedback loop

- **Continuous feedback:** Create a system for continuous feedback from both end users and stakeholders. Regularly review this feedback to make improvements and keep stakeholders informed about how their input is being utilized.
- **Suggestion forums:** Establish suggestion forums or dedicated channels where stakeholders can propose ideas and discuss potential enhancements. This fosters a sense of ownership and involvement.

Conclusion

As we conclude this journey through mastering the Software Development Life Cycle (SDLC) with low-code technology, it is clear that the integration of low-code platforms like Betty Blocks can significantly transform the way organizations approach software development. By reducing the complexity of coding and enabling rapid application development, low-code platforms empower a broader range of business users, the so-called 'Citizen Developers', to participate in the creation process, fostering innovation and collaboration.

The principles and best practices outlined in this book provide a structured approach to navigating each phase of the SDLC, from ideation to maintenance. By adopting these strategies, organizations can streamline their development processes, reduce time-to-market, and ensure the delivery of high-quality, scalable applications that meet both business and user needs.

Ultimately, mastering the SDLC with a low-code approach is not just about speeding up development; it's about building a sustainable framework for continuous improvement and innovation. With the tools and insights provided in this book, you are well-equipped to leverage the full potential of low-code technology, driving digital transformation and achieving lasting success in your software development endeavors.

About Betty Blocks

The Betty Blocks low-code platform enables organizations to build powerful solutions without being held back by technology or IT resources. With an intuitive visual interface, powerful AI capabilities, pre-built components and templates, Betty Blocks strikes the perfect balance between ease of use and functionality, making it the ideal solution for enterprise fusion teams.

These teams transform their ideas into fully customized enterprise applications. These applications, for example, automate internal processes, streamline knowledge management with AI capabilities, and engage clients with modern portals. Fully secure, governed, and scalable. As a result, organizations generate exponential growth and drive consistent innovation.



Stephan Koning
Customer Success Lead
Betty Blocks